Safety Enhancing Mechanisms for Pervasive Computing Systems in Intelligent Environments

Hen-I Yang and Abdelsalam Helal
Computer and Information Science and Engineering Department, University of Florida, Gainesville, FL 32611, USA
{hyang, helal}@cise.ufl.edu

***Abstract*—Pervasive computing systems provide personalized and intimate services to improve users' quality of life by integrating computation and communication into the environments. With the capability to interact with the physical world and the promise in assisting or managing aspects of users' daily lives, the requirement for safety is high and imminent. The difficulty in providing safety is the result of the dynamicity, complexity, heterogeneity and uncertainly typical in pervasive computing. After examining and analyzing worst-case scenarios of safety violation, we identify four fundamental elements whose individual safety assurances add greatly to the overall system safety. We propose safety enhancing mechanisms for each of the four elements and their interactions.**

I. INTRODUCTION

Pervasive computing introduces a new computation paradigm that has many unique characteristics, which presents great opportunities but also brings higher risks. The use and integration of sensors and actuators gives these systems the capabilities to interpret and influence the physical world. Many systems provide services considered to be personal, and often in private and intimate settings. Most applications are designed to be context aware and highly customizable, some even capable of predicting users' intentions and influencing their behaviors. When the computation and communication are weaved into the fabric of daily life, do users benefit from the convenience of various services, or are they being exposed to unknown risks brought upon them by these same services?

The extensive use of sensors and actuators allows pervasive computing systems to interpret and influence the physical world. They are capable of providing services and assistance by maneuvering electronic devices and physical objects such as appliances, robots or even dumb objects like doors. To the aging elderly population and persons with disabilities, these services mean critical assistance and extended independence; to others, they represent a new level of convenience. However, the capability to interact and influence the physical world implies that the impact of malfunctioned systems is no longer limited to loss of data, waste of time and effort or damage of computing devices; the possibility that users, bystanders, or physical properties can be harmed is as real as those assistance and convenience.

Many pervasive computing systems are intimate [1], because they have extensive interactions with users, and perform personal tasks on behalf of them. Intelligent environments such as smart homes [2] and intelligent vehicles [3, 4] keep extensive records of their users and act upon their preferences and limitations. These systems exist in homes and cars, which are considered to be private, safe and treated with affections. Many services involve our daily routines, including medicine reminder, scheduler, or even personal hygiene [2]. They are integrated into users' daily lives, and users have high expectations and place trust on them. Failure to perform or even causing damages can invoke strong emotions.

The semi-sentient nature of pervasive computing systems allows them to be aware of contexts, to interpret and predict users' intentions, and to extensively collect fine-grained user information. How do we ensure such systems would not betray the trust placed by their users? The famed "three laws of robotics" by Isaac Asimov provides a glimpse to the answer. Substituting robot with pervasive computing system, the first two laws can be restated as follows,

1. A pervasive computing system may not injure a human being or, through inaction, allow a human being to come to harm.
2. A pervasive computing system must obey orders given to it by human beings except where such orders would conflict with the First Law.

A pervasive computing system shares the same physical abilities as a robot in influencing and interacting with the physical world. However, robots are usually designed as stand-alone systems, while pervasive computing systems are far more user-centric and encompassing, which make these two laws even more pertinent.

Realistically speaking, it is of utmost importance to acknowledge that the impossibility to guarantee the safety of an intelligent environment. Accidents happen, people make mistakes, and Murphy's Law always looms around the corner. A glimpse of local evening news can remind us that despite best intentions and the most diligent efforts, safety may be significantly improved but the risks can never be eliminated. Weighing risks introduced and the cost to reduce or eliminate them is a continuous process.

We identify attainable goals for safety mechanisms in order of difficulties and proactivity as following:

1. Addition of new technology, devices and services should not be the primary contributor to surging risks. Similar to cost/benefit analysis for business decisions, the risk/benefit ratio needs to be assessed to ensure additional convenience significantly outweigh the extra risks introduced.
2. With capabilities to sense the environment, interact with physical world and make calm decisions, pervasive computing systems are excellent candidates to handle emergency should they occur.
3. Furthermore, we would like systems to proactively detect, prevent and manage existing risk factors.

For example, a house can catch fire if an absent-minded person leaves the stove on and forgets. A deployed pervasive system should not increase the chance of fire

0-7695-3113-X 2008

DOI 10.1109/PERCOM.2008.114

because of improper implementations or invalid operations. Actually it is much preferred if the system could monitor for and eliminate potential causes of fire hazards, or at least respond quickly should a fire starts.

Before safety mechanisms can be established, it is essential to pinpoint the potential safety hazards by examining realistic worst-case scenarios in an intelligent environment. In section 3, we identify and analyze four fundamental elements of pervasive systems and devise safety mechanisms based on these findings. Related work is presented in section 4, followed by the conclusion.

II. SCENARIOS AND ANALYSIS

A. Scenarios

1) Conflictory usage of shared resources: The lamps start to turn themselves on and off randomly like a haunted house ever since the new energy saving service is introduced on top of the exsiting lighting control service.

2) Invalid operational directives: The meal preparation service follows the exact steps in the recipe and gives instructions when the time comes to turn or add additional ingredients, and automatically shutoff to prevent overcooking. But because of a simple typo, it mistakenly set the target temperature to 3500℉ instead of 350℉ and turns the oven into an incinerator.

3) Risks of conflicting side effects: The refrigerator is not closed properly and the cold air spills onto the stove right next to it. To make sure the stew is well heated, the meal preparation service turns up the heat. To maintain constant temperature inside the refrigerator to prevent food from spoiling, the compressor amps up the power. Before long, both have been drawing so much power and a fuse is blown and the house sinks into darkness.

4) Violation of user centric computing: When he wakes up in the morning, the alarm is blaring, the bed is shaking, and his head about to explode. But the morning call service decides extra push is what it would take this morning. He struggles to climb out of the bed and turns off the alarm, and feels like screaming and throwing up at the same time. He vows that there shall never be any "smart" thing in the house from this day on.

B. Analysis

Service oriented architecture (SOA) is widely adopted by pervasive computing systems, with which each individual software or service focuses on accomplishing its own predefined goals. While it is effective in handling the dynamicity, heterogeneity and complexity of the environment, compatibility issues between services do arise, especially when services depend on one another, interact, or compete and race for shared resources. In a traditional computing system such as PC or server, the operating system monitors the use of shared resources and keeps track of the dependency. In pervasive computing systems, the lack of reliable monitoring and arbitration mechanism to oversee the large number of shared heterogeneous resources presents a major problem. The flickering lights exemplify this issue when uncoordinated energy saving and lighting services each unaware of the other's operation on the shared resource (the lamp), and issue conflicting directives.

Even when there is no shared resource involved, the side effects (also known as environmental effects) of the actions can still cause conflicts between independent services. While it is tedious and costly for each service to specify how to resolve contention for shared resources, it may not even be possible to account for all possible interferences caused by side effects. The unintentional side effects are rather limited in traditional computing, but can be a major source of risks in intelligent environments. The conflict between the fridge and stove in achieving their respective predefined goals exemplifies this issue.

Every device has its own supported operations, and the appropriate conditions under which the operations are to be performed. Improper operations and errors made by human operators usually are the result of carelessness, ignorance or misunderstanding. A well-designed device would embed hints and cues to minimize the occurrence of human errors. For instance, the temperature ranges on a thermostat and on the dial of an oven inform users the limits of their operations. However, when operated by computers, these cues, hints and visual feedbacks would fall to deaf ears, resulting in more frequent errors and worse damages. Without proper assistance and restraints, programmers who are usually offsite, become more detached from the devices, hence ignoring the visual cues and physical feedbacks that might have signaled a mistake. The software that controls devices can also receive unreasonable commands because of an error as simple as a typo. The incineration of the steak is the result of an erroneously added 0 in the cooking instruction.

Context awareness is central to many pervasive computing systems. Context reflects the condition of the surrounding environment and summarizes users' current status and intentions. When dealing with user related contexts, it often involves proper interpretations or even predictions of users' intentions. However, even human beings cannot always read each other's intentions correctly, let alone asking a computing system to do so. The misinterpretations, however, may cause serious harm and injury to users, such as the case when morning call service does not properly recognizes Jason's physical condition and worsens his headache.

C. Discussion

Similar to binding and parallelism, how and where to implement and enforce safety presents an interesting tradeoff between flexibility and efficiency. On one hand, middleware is widely employed in complex pervasive computing systems. It is the nexus of information that presents a great opportunity to intercept, examine and intervene with the data coming in and commands going out. Middleware is effective in detecting the risks resulting from dynamic interactions and exceptions arose during operations. On the other hand, the overall safety and quality of the system can be greatly enhanced if safety is incorporated in the development process. Proper APIs and programming tools can introduce and validate that appropriate measures are in place to provide certain level of assurance. Compile time support is more effective at defining exception handling routines, enforcing event driven program model, prioritizing operations and alignments, or specifying impermissible contexts.

Safety issues in pervasive computing are broad and complex therefore a collaborative effort between the middleware at run time and the programming aid at

implementation time is necessary. The middleware handles the enforcement of safety principles and dynamic behaviors, while the programming tool ensures the target systems support safety API and eliminates statically determinable unsafe operations.

III. Safety Mechanisms for Pervasive Computing Systems

A. Applicable Existing Safety Mechanisms

How do we make an intelligent environment safer? Safety is not a new concept, and many existing mechanisms are applicable to pervasive computing systems. 1. Implementing fail-safe physical safety mechanisms in addition to electric or software safety is a prudent design decision. 2. Most of the safety mechanisms for computer systems and networks such as authentication, capabilities and security protocols can be applied with little or minor modifications. 3. Since failure is the norm in pervasive computing, any mechanisms that enhance the robustness and availability to allow systems work through failures can enhance the safety of the overall system.

It is a wise decision to leave the original physical device or interfaces in place as a backup when integrating with "smart devices". For instance, to make sure the elderly is safe when the fall detection service fails to detect an emergency, it is better if the residents still carry an emergency button. To ensure the escape route is open when a fire disrupts the network or power line for the automatic door control, the regular door knob should be kept in place so people can always open the door. Using the physical device as backup in cost-effective, as these mechanisms are often already in place in regular environments. The reliability is also greatly increased because they seldom fail.

Typical safety measures for computers and networks, such as authentication, capabilities, security and privacy have all been well studied and plenty of mechanisms have been widely adopted. Applying these techniques to pervasive computing systems usually require little or minor changes. Because of the number of devices in a typical intelligent environment, and many of them are low-end sensors, failure is considered the norm during operations. Mechanisms allow dynamic brokerage [5], failure compensation using virtual sensor and service re-planning [6] to improve the availability of services all contribute to the safety of the system.

Our research effort focuses on the safety mechanisms that are specific to pervasive computing systems. The potential higher penalties associated with the risks justify designing implementation time and run time mechanisms to enhance the safety of pervasive computing systems.

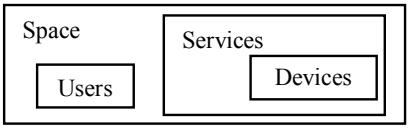

Figure 1. Four fundamental elements

B. Four Fundamental Elements

To devise effective safety mechanisms, we have to understand where the risks may arise, and what the components that need to be protected are. A quick survey of existing pervasive computing systems and an analysis of their capabilities reveal to us that there are four fundamental elements in pervasive computing systems: device, service, user and space, as shown in Figure 1.

Device: The capabilities to interpret and interact with physical world starts with devices. Whether it is a sensor, actuator or smart appliances, a device can be classified as the source of data, the recipient of commands or both. Devices can become the source of risks when they receive improper instructions or operate outside of normal operational range. Device description is used to explicitly capture or model semantic information, domain value bounds (temperature, luminance, others), operation constraints (range, maximum usage frequency, enumeration of discrete values), physical medium description, interaction protocol, and others. By providing such device description information, the middleware should be able to operate and utilize the device while enhancing the device and overall system safety.

Service: The dynamicity and heterogeneity of these systems make SOA or SOA-like architectures a prevalent practice in pervasive computing. Service is at the core of most of these systems. When employing SOA, even devices are represented as services. Services frequently exchange information and interact with others, and sometimes they can be chained or connected to create more complicated services. Each service is usually designed to fulfill very specific purposes, and has very specific requirements on the availability of other services or the existence of configurations in order to accomplish its goals. In the dynamic environment of pervasive computing, services present the smallest and simplest entity that can be well-regulated. Services can introduce risks when they violate rules on sharing resources, engage in race conditions or deadlock, or own conflicting objectives from other services. The lifecycle of services can be appropriately modeled as a state machine and their interactions can be described by interfaces.

User: Any system modeling and analysis attempts to model human behavior is laborious rarely generate any accurate results. But users are critical because they have the final say in deciding what the system need to do, and sometimes even how they should be done. Users are the single most risky factor of all, more danger is caused by the carelessness, ignorance or misunderstanding of users than problems in the system. There is not too much we can do to prevent users from doing what they do, but we can monitor users' status to keep them out of harms' way, and align systems' behavior with users' intentions as long as they are safe. User profile is designed to model users, which preferably should be both human and machine readable. User profiles are divided into two parts, the static profile, which describes users' preferences, limitations, and priorities, is relatively stable and usually not affected by the system; the dynamic profile, on the other hand, consists of the user related data gathered by the system, for instance, the current location of the user, the blood pressure and glucose level attained by medical sensors, and is continuously updated by the system.

Space: Many do not consider a space itself to be a critical element in the system. However, a space is an important element because the status of the space as a whole is critical for context-awareness; a space also encompasses all other elements within, in particular the status of services, devices and users as well as the

interactions between them; side affects usually cannot be captured by devices or services, but can be described by measurable changes in a space. Based on functions, social contexts and the locale of a space, there are different restrictions and interpretations of the contexts. For instance, the interpretation of cleanness in a clean room will be different from a butcher shop; the acceptable temperature and humidity also drastically differ in an engine room from an operation room. The differences can be defined in the context interpretation descriptions so raw data can be interpreted accordingly. The state of a space can be captured using ontology-based context graph, which not only tells the system how to interpret low-level sensor readings into contexts, but can be used to monitor and visualize the currently active contexts.

Operating in a dynamic, heterogeneous and uncertain environment, the safety issue of pervasive computing systems is complex. Since they are all composed of these four fundamental elements, any potential risks would involve at least one of them. Employing the strategy of "divide and conquer", we devise the safety mechanisms by first securing each element, and then reducing risks occurred during their interactions. Finally, since the space is the all-encompassing element, we use the safety mechanisms for space as a safety net to capture anything unintentional or fall through the crack.

C. Safety Mechanisms for Pervasive Computing Systems

1) Static safety mechanisms – Prioritzed Safety API

a) State machine and service safety interface: This is a service protection mechanism to allow smooth operations and enhance control and tracking of services in dynamic environment typical for pervasive computing, we adopt the event-driven programming paradigm, which is based on a state machine for all services as shown in Figure 2. *init* is the state when a service object first instantiated; a service is *binded* when successfully acquired all dependent services; before execution a service needs to be *aligned* with users' preferences and limitations; only then can a service start *executing*. When emergency hits, a service can move into *emergencyPowerDown* from any other state. Service safety interface is the manifestation of the transitions in the state machine. State of the service needs to be specified. Rationale too.

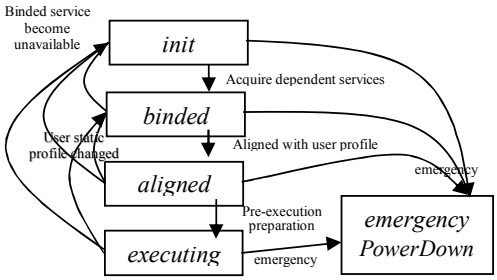

Figure 2. State diagram for services

b) Mandatory power down sequence for all service: onEmergencyPowerDown() method, part of the service safety interface, is mandatory for every service, in which programmers would describe the necessary steps, such as setting variable to a safe value (e.g. set the oven temperature to 0 °F before stopping the service), releasing any shared resource it currently holds, report power down status to service registry, and notifying services it currently collaborates with, to ensure a service can be parked safely. This sequence is required to be reliable, atomic and follows a well defined protocol for termination in case of failures.

c) Preemptive methods with priority: Methods are designated with one of the two priorities, high priority for emergency handling methods, and regular priority for all other methods. Regular priority methods should be implemented as preemptible using similar techniques as preemptive concurrent programming.

2) Dynamic safety mechanisms in the middleware: Figure 3 summarizes the dynamic safety mechanisms supported by the middleware. The details of some of these mechanisms are given in the following subsections.

a) Device safety checker: Each instruction issued to the devices needs to go through additional checking to ensure their conformance to the limitations of the target device, including the method invoked is supported, the operands are within the normal operational range, and the frequency of alternating instrucionts is acceptable. From the perspective of object-oriented programming, it is preferred these checkings are performed within the device service object, but devices with small footprint or higher code reuse, they can also be outsourced to Device Safety Checker with device id and the parameters of instructions.

b) Context manager and emegency detector: Context manager uses context graph to interpret raw sensor readings into higher-level contexts Contexts are effective ways to describe the overall status of everything in the space, including the users of the system. Active contexts are provided to services for making context-aware decisions. Emergency detector also check these active contexts to see if any dangerous *impermissible contexts* has come true. Should any impermissible context be triggered, the emergency detector would invoke the handling routine in the emergency handler vector (EHV) using the associated emergency number.

c) Emergency Handler Vector (EHV): This is the centralized emergency response routine similar to the interrupt vector. When each device and service first joins the system, they are required to deposit emegency handlers into EHV should any major problem arises; there is also a handler for each impermissible context defined in the context graph. When an emergency handler is activated, EHV issues preemptive high-priority calls to suspend services, park devices, decouple dependencies or issue overrides to reverse the emergency situation.

d) Service Registry: The service registry uses both volunteery reporting and active probes to acquire the current state of each service, as well as the depedencies established during the binding process. The purpose is to provide the system an overview of all the services in the system. The purpose of the service reigstry is to handle the situation where the service object behaves irradically or becomes irresponsive. In these situations, the middleware overtakes the control by invoking "hard park" routine,

which terminates the erroraneous service object and controls the devices or services directly. As compared to the usual "soft park" routine specified in the *onEmergencyPowerDown* () method.

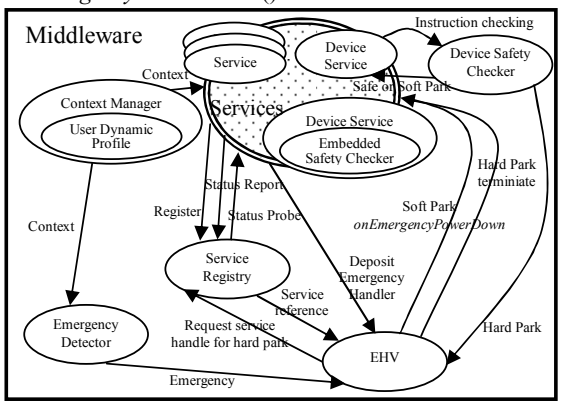

Figure 3. Dynamic safety mechanisms in the middleware

D. Effectiveness of the Safety Mechanisms

Using the safety violation scenarios identified earlier, we now demonstrate how safety risks can be mitigated or eliminated with the proposed mechanisms.

a) Initialization: A new service is introduced when a device is brought into the space or a software artifact is inserted into the system. To ensure all services exhibit safety awareness, each is required to implement the prioritized safety API. The API mandates the provision of a safe method for powering down during emergency. It also ensures other methods has lower priority, and can be preempted should emergency arise.

When a new service arrives, it needs to be initialized by depositing the emergency handler to EHV, contacting the service registry and reporting its current status. New services are not allowed to interact with others or be used to compose composite services until EHV and service registry cleared their initializations. Once verified, these services can be integrated as part of the system and start offering useful services to users.

Introduction of each sensor, actuator, and certain software artifact such as virtual sensors [6], allow the system to gain the ability to better interpret and interact in expanded dimensions. To build context-aware systems, programmers first define the mapping between the sensor readings and the basic contexts. They can further derive more abstract contexts using existing context definitions. Furthermore, dangerous situations should be defined as impermissible contexts, and corresponding emergency handlers should be created and deposited into EHV.

b) Runtime safety protection: When the lamp service receives interleaving contradictory instructions to turn on and off in a higher than acceptable frequency, the device safety checker first decides if one has higher priority than others. It would then filter and ignore the less critical ones. If they are of equal priority, the service is transited to *emergencyPowerDown* state to avoid potential safety hazards, and emergency handler can safely takes the service offline.

Similarly, services are moved to *emergencyPowerDown* state if invalid instructions arrive, such as the one requesting the oven to operate at an unreasonable 3500 °F. Once the device services are power down and taken offline, the composite services that depend on them will have to either re-compose themselves by substituting with alternative services, or to follow suit and propagate the power down. In either case, the conflicts exerted on shared resources and invalid instructions will cease to have effects and potential risks averted.

When cold air spills, the interaction between the fridge and the stove cannot be easily foreseen, underscoring the difficulty in proactively define and defend against all possible hazards resulting from combinatory side effects. The improperly closed door of the fridge ends up causing unsafe power draw and blown fuse. This type of hazards cannot be readily handled by device safety checker. Instead, the emergency detector works in tandem with context manager, constantly monitors the overall status of the space and users within. The emergency detector does not proactively prevent the unsafe situation from emerging, and is not associated with any specific service or device, it simply monitors and reacts when safety hazards occur. By defining extreme power draw as an impermissible context, and specifying an emergency handler to prevent any additional request for more power, and prioritize the existing power-drawing services, the system can prevent and control catastrophic safety hazards.

Successful pervasive computing systems in intelligent environments have to be user centric. Many projects strive for context awareness, which is important, but we argue that all services and the space as a whole have to focus on what are beneficial to users and respect their preferences and limitations. The explicit *aligned* state for services enforces this notion. By ensuring each service is aligned with users' profiles before execution, programmers are forced to consider users during the design and implementation process. If aligned with the user's dynamic profile, the morning call service would detect abnormal context of sickness, and instead of making him sicker by rushing him out of the bed, the aligned service would retrieves the contact of physician from the EHV and initiates a remote-care communication.

If the service in question hogs resource, executes non-preemptible methods, or becomes unresponsive, the emergency detector will look up the emergency handler in EHV, get a handle on the service from registry and kills the service instance. It will perform follow-up procedures to make sure the share resources are released, and the devices are placed in a safe setting before shutdown.

c) Discussion: The proposed mechanisms provide basic safety protection against abuse of devices and services, ensure explicit user profile alignment, and use impermissible contexts as the second line of defense against safety risks not explicitly tied to any particular service or device. They provide invaluable safety enhancements over existing pervasive computing systems, but does not offer guarantee on safety. Just like any product or system that claim to be safe, such as a safe razor, the term "safe" is only used to describe that the product is safer than alternatives [8].

These mechanisms promote the safety awareness during the design and implementation process. It prompts programmers to consider safe emergency power down sequence, the normal operation conditions for each device

and service, and how users' conditions, preferences and limitations affect the operation of services. However, the responsibility of creating safe systems still lies with designers and programmers. They can implement an empty emergency handler without any instruction, or create align method without actually consider users' profiles, or not specifying acceptable operation conditions of devices. Procedures ineffective against safety hazards might also be included. Due to the heterogeneity of devices and services, and dynamicity in terms of possible service combinations, coupled with potential risks caused by users, system and the nature, or their combinations, there is really no way to ensure effective handling against all potential safety risks. The proposed mechanisms provide effective safeguard against safety hazards, assuming programmers diligently implement the methods in the API with effective safety-aware operations.

Just like any engineering disciplines, the proposed safety mechanisms are designed to protect against certain safety risks already identified, such as ones in the scenario. The difference is that pervasive computing has only recently emerged in the past two decades, and lacks mass deployments in the real world environments. There are many potential safety risks still yet to be identified. One of our ongoing works is to further explore other safety hazards and design mechanisms to counter them.

Despite the fact that most pervasive computing systems are general purpose, making it impossible to devise exhaustive protections against specific safety threats, the aforementioned mechanisms offer effective protections against devise and space related risks. We are currently looking to strengthen protection on services with risks associated with shared resources and resource usage.

IV. Related Work

There are limited studies on the safety issues of pervasive computing systems. A compile time semantic checking mechanism on a functional/logic programming language is proposed for implementation and verification of safe pervasive computing systems [7]. But the result cannot be directly applied to systems implemented using more popular languages, as is the case for most systems implemented. Leveson uses software engineering practice to devise safety modeling using fault tree and probability analysis, and identifies requirements and design for safety [8]. Her model and practice help to shape our solution, but do not provide any immediately applicable design specifics for pervasive computing systems.

Other studies address safety issues in particular application domains, such as intelligent vehicles [3, 4] and smart hospital [9]. Some focus on human factors and usability when delivering information [10], others on formal methods and the software engineering practice in designing and certifying the safety of distributed systems used in vehicles [11] and driver vigilance monitoring [12]. Many projects have applied non-functional requirements of security, privacy, usability and reliability (SPUR) in the automotive context. Although all the requirements have direct or indirect impacts on the safety of drivers and vehicles, the main focus of SPUR is not on making these systems safer. Our goal is to create generic safety mechanisms applicable to a wide range of pervasive systems in intelligent environments.

Many middleware have been designed for pervasive computing [9, 13], but they have other primary design goals such as quality of contexts and ease of integration. None was designed to address the safety issue.

V. Conclusion

Safety is defined as "the condition of being safe from undergoing or causing hurt, injury or loss". For pervasive computing systems, especially those deployed in intelligent environments, safety should be as important as effectiveness or usability. The ability to influence the physical world and the often intimate interactions with users means any unsafe operation would cause more drastic physical and psychological damages. Even with all the assistance and convenience that come with these systems, there should be no unreasonable or unnecessary increase in risks to the well-being and property of users.

The dynamicity, complexity, heterogeneity and uncertainty of pervasive computing systems pose serious challenges in guaranteeing safety. The lack of reliable monitoring and arbitration mechanisms such as the ones provided in operating systems in traditional computers contributes to the problem. The identification, analysis and modeling of the four fundamental elements, which are devices, services, users and space, shed some lights on how to make these systems safer. Securing the safety of each and managing the risks of their interactions provides a promising approach in making pervasive computing systems in intelligent environment a safer endeavor.

References

[1] G. Bell, "Intimate Computing?" *IEEE Internet Computing*, vol. 8, issue 6, pp. 93 – 93, November 2004.

[2] A. Helal, W. Mann, H. Elzabadani, J. King, Y. Kaddourah and E. Jansen, "Gator Tech Smart House: A Programmable Pervasive Space," *IEEE Computer magazine*, pp 64-74, March 2005.

[3] TJ Giuli, D. Watson and K. Prasad, ""The Last Inch at 70 Miles Per Hour," *IEEE Pervasive Computing Magazine*, vol. 5, number 4, pp. 20 – 27, Oct. – Dec. 2006.

[4] Y. Liu, and Z. Wu, "Comfortable driver behavior modeling for car following of pervasive computing environment", Computational Science – ICCS 2005, Springer-Verlag, pp. 1068 – 1071, 2005.

[5] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, and K. Nahrstedt, "Gaia: a middleware infrastructure to enable active spaces," *IEEE Pervasive Computing*, pp. 74-83, 2002.

[6] J. Xia, C. Chang, T. Kim, H. Yang, R. Bose, and A. Helal, "Fault-resilient Ubiquitous Service Composition", *Proceedings of IE'07*, Ulm, Germany, pp. 108 -115, Sept. 2007.

[7] DSSE Group in University of South Hampton, "Safety in Pervasive Computing," Advanced Specialization and Analysis for Pervasive Computing project deliverable report D13, 2003.

[8] N. Leveson, *Safeware: System Safety and Computers*, Addison Wesley, 1995, pp. 313-446.

[9] J. Bohn, F. Gärtner and H. Vogt, "Dependability Issues of Pervasive Computing in a Healthcare Environment," *LNCS 2082*, Springer-Verlag, pp. 53-70, 2003.

[10] P. Green, "Driver Interfaces for Vehicles of the Future and Usability Roadblocks to Their Introduction," *Inside Automotives*, vol. 3, number 6, pp. 33-36, 1996.

[11] K. Tindell, H. Kopetz, F. Wolf and R. Ernst, "Safe automotive software development", *Proceedings of DATE '03*, 2003.

[12] J. Qiang, Z. Zhiwei and P. Lan, "Real-time nonintrusive monitoring and prediction of driver fatigue", *IEEE Transactions on Vehicular Technology*, vol. 53, issue. 4, pp 1052-1068, 2004.

[13] J. King, R. Bose, H. Yang, S. Pickles and A. Helal, "Atlas – A Service-Oriented Sensor Platform," *Proceedings of SenseApp2006,* November 2006.